A Parallel Algorithm for Inheritance Hierarchies with Constraints

AUTHOR(S):

Menju, Satoshi; Itoh, Hidenori; Morita, Yukihiro

CITATION:

Menju, Satoshi ...[et al]. A Parallel Algorithm for Inheritance Hierarchies with Constraints. 数理解析研究所講究録 1989, 709: 15-32

ISSUE DATE:

1989-12

URL:

http://hdl.handle.net/2433/101663

数理解析研究所講究録
第709巻 1989年 15-32

15

A Parallel Algorithm for Inheritance Hierarchies with Constraints

Satoshi Menju, Hidenori Itoh and Yukihiro Morita

ICOT Research Center
21F, Mita Kokusai Bldg.,
1-4-28, Mita, Minato-ku, Tokyo, 108, Japan

Abstract

This paper introduces the concept of constraints in multiple inheritance hierarchies with exceptions. We add constraints to the links of inheritance networks, then consider links with satisfied constraints only, ignoring links with unsatisfied constraints. This method can increase the expressive power.

This paper also describes a parallel algorithm for inheritance hierarchies with constraints. It terminates in $O(n)$-time, where n is the length of the longest path of an inheritance network including constraints. The algorithm obtains one of the solutions produced by the credulous reasoners of Touretzky and Etherington. We also implement the algorithm in a parallel logic programming language, Guarded Horn Clauses (GHC).

1. Introduction

Algorithms to process multiple inheritance with exceptions have been discussed in [1,2,3,4]. In these algorithms, the property inheritance between two objects is represented statically by a link. This paper introduces the idea of constraints to the representation of more complex multiple inheritance problems. The idea of constraints has been argued in logic programming languages [5,6] to solve more complex logical problems. Constraint logic programming languages are also knowledge representation languages using constraints.

From the point of view of constraints and their processing in parallel, the parallel algorithm presented in this paper is more powerful than the previous ones. It is expected to be a new paradigm of knowledge representation or knowledge management language.

In this research area, both representation power and processing efficiency are important. This paper explains firstly, the usual inheritance network model, secondly, our model whose link contains added constraint information, and thirdly, an efficient parallel algorithm to process it. The algorithm is written in Guarded Horn Clauses (GHC) [7], a parallel logic programming language defined as the kernel language of the Fifth Generation Computer System project in Japan.

2. Inheritance Network

This section lists some basic preparations for the following discussion. First, a usual multiple inheritance network is defined. Suppose that there are a set of individuals and a set of predicates.

Definition 2.1 $< x, +y >$ and $< x, -y >$ are *inheritance links* (or, more simply, *links*), where x is an individual or a predicate and y is a predicate.
$< x, +y >$ is called an *is-a* link, and describes that x is y. Similarly, $< x, -y >$ is called an *is-not-a* link, and describes x is not y.

Definition 2.2 When there is a link from x to y, y is called a *parent* of x and x is called a *child* of y.

Definition 2.3 An *inheritance network* consists of a set of individuals, a set of predicates, and a set of links.

In this paper, we assume that an inheritance network includes at least one link from any node to another node.

Graphically, an individual is denoted by a white node, a predicate is denoted by a black node, an *is-a* link is denoted by an arrow and an *is-not-a* link is denoted by a crosshatched arrow.

Definition 2.4 If there are links $< x_i, +x_{i+1} >$ or $< x_i, -x_{i+1} >$ for $1 \leq i \leq n-1$, then $[x_1, ..., x_n]$ is called a *path*. When there are links $< x_i, +x_{i+1} >$ for $1 \leq i \leq n-1$, $1 \leq n$, if there is a link $< x_n, +y >$ then the path $[x_1, x_2, ..., x_n, y]$ is called a *positive path*, if there is a link $< x_n, -z >$ then the path $[x_1, x_2, ..., x_n, z]$ is called a *negative path*. Any other path is a *meaningless path*.

Definition 2.5 An *is-a* link is a *positive reasoning path*, and an *is-not-a* link is a *negative reasoning path*. When there is a positive reasoning path from x_1 to x_n and an *is-a* (*is-not-a*) link from x_n to y, if there is no possibility of a negative (positive) path from x_1 to y, then the positive (negative) path from x_1 to y is called a *positive* (*negative*) *reasoning path*. If both a positive path and a negative path are possible, then we use the inferential distance ordering by Touretzky (see [1]) to choose the reasoning path.

Touretzky's ordering is essentially that if there is a path from x to z through y, then y is considered *nearer* to x than z is to x. If both a positive path and a negative path are possible, then the path whose last link leaves the child node nearer to the starting node of paths, is preferred, and is considered as a reasoning path.

Now we assume two restrictions on our model as follows.

(1) Every pair of nodes has at most one link.
(2) No path makes a loop.

These restrictions are reasonable, because to have the same kind of links between a pair of nodes is redundant, and to have both kind of links between a pair of nodes would lead to contradiction between the links. In inheritance hierarchies, a link means "including", so it is not necessary to consider any loops.

Here, a positive node set, a negative node set and a non-conclusion node set are defined.

Definition 2.6 A *positive node set of the node s* is a set of nodes to which there is a positive reasoning path from s. A *negative node set of the node s* is a set of nodes to which there is a negative reasoning path from s. A *non-conclusive node set of the node s* is a set of nodes that belong to neither the positive node set nor the negative node set of the node s.

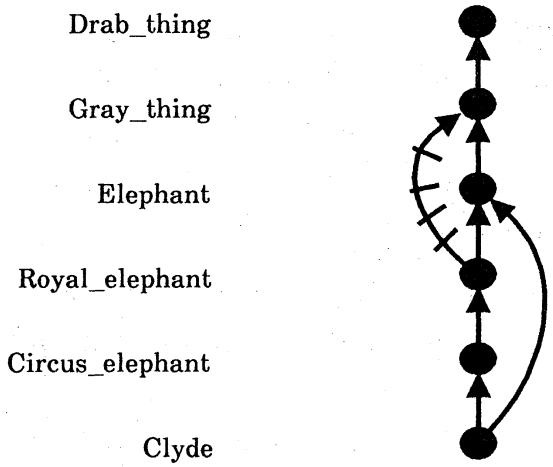


Figure 1 Description of Clyde

Definition 2.7 A triplet (*positive node set of s, negative node set of s, non-conclusive node set of s*) is a *resolution for s.*

In this paper, we consider the problem that given an inheritance network and a question node s, we obtain the resolution for s.

Here is an example [1]. In Fig. 1, P_1 = [*Clyde, Elephant, Gray_thing*] is a positive path, P_2 = [*Clyde, Circus_elephant, Royal _elephant, Gray_thing*] is a negative path, and [*Clyde, Circus_elephant, Royal_elephant, Gray_thing, Drab_thing*] is a meaningless path. The node *Royal_elephant* is nearer to *Clyde* than *Elephant.* So P_2 is a reasoning path, and P_1 is not. The resolution for *Clyde* is ({*Circus_elephant, Royal_elephant, Elephant*}, {*Gray_thing*}, {*Drab_thing*}).

The inferential distance ordering is not tatally order. So there are cases in which there are many resolutions for a node in an inheritance network. Then there are two ways to present the resolutions. One is credulous reasoning which gathers all possible resolutions. The other is skeptical reasoning in which, if after using the inferential distance ordering both a positive reasoning path and a negative reasoning path to x are possible, no path to a node x is considered as a reasoning path [1,2].

We have already shown a parallel reasoning algorithm to find one of the resolutions using credulous reasoning, and have also shown that its time complexity is $O(n)$, where

n is the length of the longest path in the inheritance network [4].

In the next section, we extend inheritance network by introducing constraints.

3. Inheritance Network with Constraints

This section discusses an inheritance network with constraints.

Suppose that conditions such as times and places are decided to be true, false or unknown. A link with constraints is defined as follows.

Definition 3.1 Let c be a condition. A *constraint* is the form $(+, c)$ or $(-, c)$. $(+, c)$ is satisfied when c is true, while $(-, c)$ is satisfied when c is false or unknown.

Definition 3.2 An *is-a (is-not-a) link from p to q with a set, C, of constraints* is represented by $< p, +q, C >$ ($< p, -q, C >$). When all constraints in C are satisfied, p can be reasoned to be (not to be) q. When at least one of the constraints in C are not satisfied, p cannot be reasoned to be (not to be) q using this link.

We represent predicates by nodes in inheritance networks. Conditions are predicates, so we identify conditions by nodes that correspond to the conditions. In inheritance networks, we represent constraints by links from nodes that correspond to the conditions to inheritance links with the constraints.

Definition 3.3 We call links, representing constraints, *constraint links*. $(+, c)$ is represented by a *positive constraint link* $- - - >$ from node c. $(-, c)$ is represented by a *negative constraint link* $+ + + >$ from node c.

Definition 3.4 Paths are *physical paths*. When a link from node a to node b has a constraint from node c, if there are physical paths $P1 = [x_1, ..., x_n(=\mathrm{c})]$ and $P2 = [y_1(= b), ..., y_m]$, we consider that there is a *physical path* from x_1 to y_m through $P1$, the constraint link and $P2$, and we represent the physical path by $[x_1, ..., x_n, (a), y_1, ..., y_m]$. The length of the physical path is $m + n - 1$.

When we want to discover the nodes from which a question node inherits properties, we must give some information for constraints in addition to a question node.

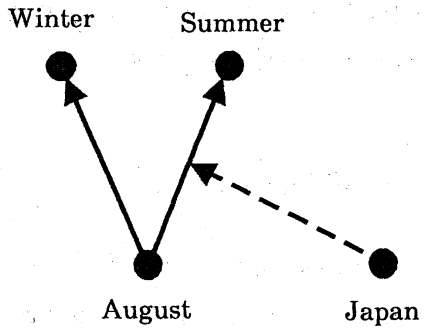


Figure 2 Simple example

Definition 3.5 If the information "a node is true (false)" is given, we call the node a *designated node as true (false, respectively)*. Moreover, if a node is a question node or a designated node as true or false, then we call the node a *starting node.*

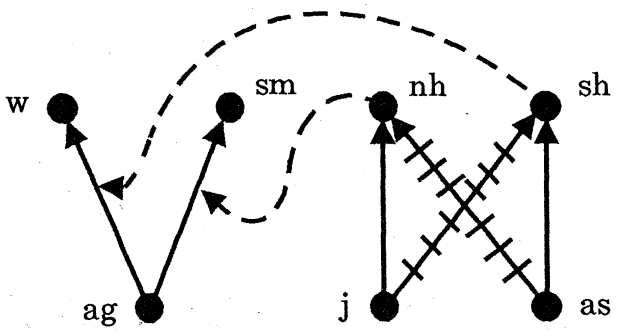
Fig. 2 shows a simple example. The link from the node *August* to the node *Summer* has a positive constraint from the node *Northernhemisphere.* When a question node *August* and an information "northernhemisphere is true" are given, the positive constraint is satisfied, and "August is Summer" is reasoned. There is a physical path [*Northernhemisphere,* (*August*),*Summer*], and its length is 1.

To give all conditions that are true is inefficient and redundant. It is better to reason whether some conditions are true or not.

Definition 3.6 If a node (predicate) belongs to the positive node set of a question node or of a designated node as true, and if it is not a designated node as false, then it is regarded as true.

Here, the previous example is extended (Fig. 3). Suppose that *August* is given as the question node, and *Japan* as the designated node as true. Since the positive node set of the node *Japan* is {*Japan, Northernhemisphere*}, *Northernhemisphere* is true. Then the constraint of the link from *August* to *Summer* is satisfied, and the constraint of the link from *August* to *Winter* is not satisfied. So only "August is Summer" is reasoned.

Next, some functions that can be represented by using constraints are shown.



ag : August	sm : Summer	w : Winter
j : Japan	nh : Northernhemisphere	
as : Australia	sh : Southernhemisphere	

Figure 3 Extended example

(1) Priority

When there are $< a, +c, \{(-, b)\} >$ and $< a, +b, \{\} >$ (Fig.4-a), you can generally reason that a thing belonging to a belongs to b. But if you know that one thing belonging to a does not belong to b, you can reason that it belongs to c.

(2) Exclusive

$< a, +b, \{(-, c)\} >$ and $< a, +c, \{(-, b)\} >$ mean that a can be reasoned to be b or c exclusively. This is shown in Fig. 4-b.

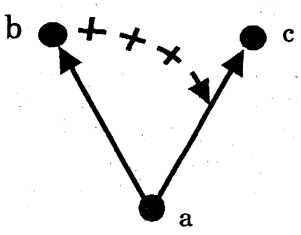
(3) Default (Etherington and Reiter [3])

Etherington and Reiter's example [3] is written by the following links with constraints (Fig. 4-c).

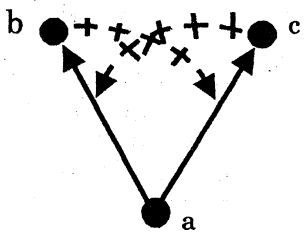
$$< n, +s, \{\} >, \quad < n, +c, \{\} >, \quad < c, +m, \{\} >,$$
$$< c, -s, \{(-, n)\} >, \quad < m, +s, \{(-, c)\} >$$

4. Parallel Algorithm

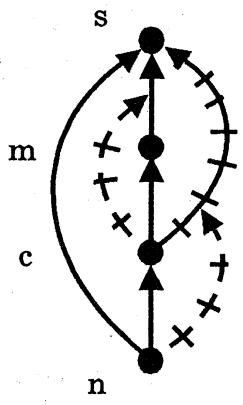
This section shows a parallel algorithm to find one of the resolutions by credulous reasoning. Before showing it, we impose the following restriction on the inheritance networks.



(a) Priority



(b) Exclusive



s	**: Shell-bearer**
m	**: Mollusk**
c	**: Cephalopod**
n	**: Nautilus**

(c) Default

Figure 4 Functions

No physical path makes a cycle.

The example in Fig. 4-*b* has a cycle of paths $[b, (a), c, (a), b]$. Our parallel algorithm treats only acyclic inheritance networks. The algorithm can be easily improved to treat some inheritance networks with cycles.

The algorithm puts one of three kinds of markers on each node in the inheritance network. This resolves the resolution. Next we define markers.

Definition 4.1 Markers we use in the algorithm are *tm*, *fm* and *mm*. Nodes with *tm* (*fm, mm*) belong to the positive (negative, non-conclusive, respectively) node set.

Given a question node and a set of designated nodes as true or false, first we put appropriate markers on them and propagate markers to other nodes through links. At that time, the algorithm also propagates the logical path lengths and priorities together markers.

Definition 4.2 The *logical path length* of a starting node is 0. The *logical path length* of another node is the maximum length of paths through which *tm* or *fm* is propagated from a starting node to the node.

The *priority* is given with a marker, and propagated with the marker. When markers propagated from different starting nodes to a node conflict with each other, the preferred marker is decided by using their priorities.

We use natural numbers greater than 0 to represent priorities. The lower the number, the higher the priority.

The algorithm actually puts a triplet $(mark, p, l)$ on each node, where $mark$ is a marker, p is a priority, and l is a logical path length.

Next, the marker propagation rule is defined as follows.

Suppose that node x has a triplet (tm, p, l), and there is an is-a (is-not-a) link from x to node y. If all constraints of the link are satisfied, the triplet $(tm, p, l+1)$ $((fm, p, l+1)$, respectively) is propagated from x to y. Otherwise the triplet (mm, ∞, ∞) is propagated from x to y. See Fig. 5.

In this way, these triplets are propagated in parallel to every parent of the node whose triplet is known. Note that if a triplet has an *mm*, then both the logical path length and the priority are ∞.

<table>
<tr><td></td><td></td><td>(tm, p, l)</td><td>(fm, p, l)</td><td>(mm, ∞, ∞)</td></tr>
<tr><td rowspan="2">satisfied</td><td>is-a</td><td>$(tm, p, l+1)$</td><td colspan="2" rowspan="2">(mm, ∞, ∞)</td></tr>
<tr><td>is-not-a</td><td>$(fm, p, l+1)$</td></tr>
<tr><td>unsatisfied</td><td>all</td><td colspan="3"></td></tr>
</table>

Figure 5 Propagation rule

Next, we show the algorithm itself. The input is a list of triplets ($node_i$, $mark_i$, $priority_i$), where $node_i$ is a starting node, $mark_i$ is a marker put on $node_i$, and $priority_i$ is the priority of $mark_i$. The question node is distinguished from designated nodes by their priorities: the marker on the question node generally has the highest priority, 1.

Step 1. Put the triplet $(mark_i, priority_i, 0)$ on every $node_i$ in the input list.

Step 2. For every node that has a triplet (m, p, l), and that has a link to node b which does not have a triplet, in parallel, the following are executed.

(1) If the link is *is-a*, all constraints of the link are satisfied, and m is tm, then send the triplet $(tm, p, l+1)$ to b.

(2) If the link is *is-not-a*, all constraints of the link are satisfied, and m is tm, then send the triplet $(fm, p, l+1)$ to b.

(3) If some constraints of the link are unsatisfied, or if m is not tm, then send the triplet (mm, ∞, ∞) to b.

Step 3. For every node that receives triplets (m_i, p_i, l_i) from all child nodes, in parallel, the triplet (M, P, L) is put, where M, P, and L are defined as follows.

$P = p'$, where p' is a minimum value of p_i.

$L = l'$, where l' is a maximum value of l_i whose priority is p'.

$M = m'$, where the triplet (m', p'', l'') exists, and l'' is a minimum value of l_i' whose priority is p'. m' is a marker propagated through the path of the minimum logical path length.

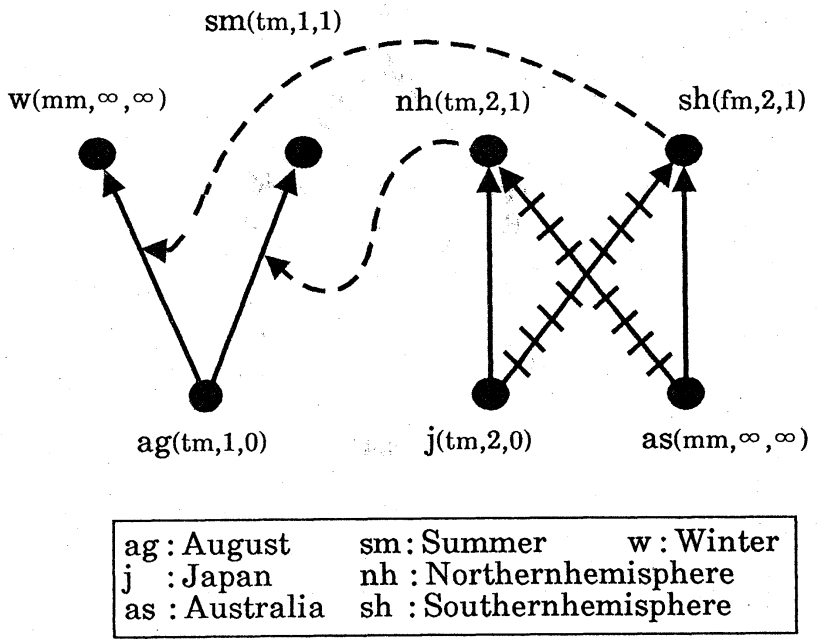


Figure 6 Example of a parallel algorithm

Step 4. If there are nodes on which triplets are put newly in Step 3, then go to Step 2. If not, put a triplet (mm, ∞, ∞) on each node that has no triplet, then terminate.

Note that there are cases in which M can be both *tm* and *fm*. In these cases, whichever marker is selected, the solution by our algorithm is one of the resolutions by credulous reasoning. At the beginning of processing, the user can decide which marker, *tm* or *fm*, is selected in those cases, according to the property of the given problem.

The extended example explained in section 3 is shown in Fig. 6 again. We want to find an answer to the following question.

Is it summer in August in Japan?

In this example, *August* is a question node and *Japan* is a designated node as true. Initially, the node *August* has a triplet, $(tm, 1, 0)$, and *Japan* has a triplet, $(tm, 2, 0)$. Using the parallel algorithm, we can find the answer node, *Summer*, with $(tm, 1, \_)$ attached.

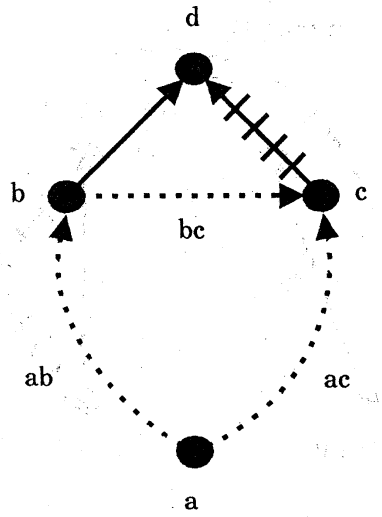


Figure 7 Preference of Path

Because our inheritance network is restricted to being acyclic and has a finite number of nodes, the following theorems are obvious.

Theorem 1 Our parallel algorithm obtains one of the resolutions by credulous reasoning.

By using the example (see Fig. 7), we show the main idea of the proof. Suppose that there are a positive path ab from a node a to a node b, a positive path ac from a to a node c, a positive path bc from b to c, an is-a link from b to d, and an is-not-a link from c to d. In credulous reasoning, the positive path abd from a to d through the is-a link from b to d is preferred to the negative path acd from a to d through the is-not-a link from c to d. Let the logical path length of a path P be denoted by $lpl(P)$. Then,

$$lpl(abd) = lpl(ab) + 1,$$

$$lpl(acd) = \max\{lpl(ac),\ lpl(ab) + lpl(bc)\} + 1.$$

Hence,

$$lpl(acd) \geq lpl(abd).$$

So, in our algorithm, the marker propagated through the path abd is put on the node d

Theorem 2 The time complexity of our parallel algorithm is $O(n)$, where n is the maximum length of the physical paths in the inheritance network.

It is obvious, because Step 2 through Step 4 are executed n times.

5. Programs in Parallel Logic Programming Language GHC

This section shows programs in parallel logic programming language Guarded Horn Clauses (GHC) [7] according to the parallel algorithm given in the previous section.

The structure of Fig. 3 is written as shown in Fig. 8 (b) in GHC. For comparison, a program with neither constraints nor priorities is shown in Fig. 8 (a). It cannot find a correct solution in this case. *gen* and *node* are processes defined in GHC. *Ms*, which is the input of the *gen* process, is a list of the triplets ($node_i$, $mark_i$, $priority_i$). N, which is the output of the *gen* process is a list of pairs of the marker and priority of each node.

The first argument of the *node* process is the name of the node in Fig. 3. The second argument is an input list of the *gen* process. The third argument is the marker given at the node. The fourth argument is the priority given at the node. The fifth argument is a list of pairs of markers and constraints at the node. The sixth argument is a common variable defined in GHC; it is used for the communication with the parent nodes connected by *is-a* links. The seventh argument is also a common variable; it is used for communication with the parent nodes connected by *is-not-a* links.

Each process *node* receives a triplet (*marker, priority, logical path length*) from the process *node* connected with the common variable, calculates its own triplet, and then sends them through the common variable in the sixth and the seventh arguments.

The appendix gives the whole program in GHC.

6. Conclusion

This paper introduced the idea of inheritance network with constraints and a method of representation. This method of representation is expected to become a useful knowledge representation language, because, in many cases, the relations between objects represent constraints.

```
gen(Ms,N) :- true |
        node(ag,Ms,Na,[]        ,TMa,_  ),
        node(sm,Ms,Nb,[TMa]     ,_  ,_  ),
        node(w ,Ms,Nc,[TMa]     ,_  ,_  ),
        node(j ,Ms,Nd,[]        ,TMd,FMd),
        node(nh,Ms,Ne,[TMd,FMf],_  ,_  ),
        node(as,Ms,Nf,[]        ,TMf,FMf),
        node(sh,Ms,Ng,[TMf,FMd],_  ,_  ),
        N=[Na,Nb,Nc,Nd,Ne,Nf,Ng].
```

(a) Program for Fig. 3 without constraints

```
gen(Ms,N) :- true |
        node(ag,Ms,Na,Pa,[]                    ,TMa,_  ),
        node(sm,Ms,Nb,Pb,[(TMa,[(+,Ne)])]      ,_  ,_  ),
        node(w ,Ms,Nc,Pc,[(TMa,[(+,Ng)])]      ,_  ,_  ),
        node(j ,Ms,Nd,Pd,[]                    ,TMd,FMd),
        node(nh,Ms,Ne,Pe,[(TMd,[]),(FMf,[])],_  ,_  ),
        node(as,Ms,Nf,Pf,[]                    ,TMf,FMf),
        node(sh,Ms,Ng,Pg,[(TMf,[]),(FMd,[])],_  ,_  ),
        N=[(Na,Pa),(Nb,Pb),(Nc,Pc),(Nd,Pd),(Ne,Pe),(Nf,Pf),(Ng,Pg)].
```

(b) Program for Fig. 3

Figure 8 Sample program in GHC

A parallel algorithm for inheritance networks with constraints is also shown. This algorithm seems to be an expansion of Touretzky's algorithm for static multiple inheritance without constraints.

This algorithm terminates in $O(n)$-time, where n is the maximum length of the physical paths in the inheritance network, because our inheritance networks have no cycle.

This paper discussed the model and approaches to parallel processing, co-operative problem solving and logic programming with constraints. The feasibility and flexibility of the parallel algorithm must be verified by applying it to larger real applications.

References

[1] Touretzky, D. S., *The Mathematics of Inheritance Systems*, Morgan Kaufmann Publishers, Los Altos, CA, 1986.

[2] Horty, J. F., Thomason, R. H. and Touretzky, D. S., A Skeptical Theory of Inheritance in Nonmonotonic Semantic Networks, *Proceedings of AAAI-87*, 1987, pp.358-363.

[3] Etherington, D. W. and Reiter, R., On Inheritance Hierarchies With Exceptions, *Proceedings of AAAI-83*, 1983, pp.104-108.

[4] Menju, S., Morita, Y. and Itoh, H., A Parallel Algorithm for Inheritance Hierarchies with Exceptions, *Proceedings of 36th IPSJ Conference 6P-8*, 1988, pp.1491-1492 (in Japanese).

[5] Colmerauer, A., Opening the Prolog III Universe: A New Generation of Prolog Promises Some Powerful Capabilities, *BYTE*, August 1987, pp.177-182.

[6] Jaffar, J. and Lassez, J-L., Constraint Logic Programming, In *4th IEEE Symposium on Logic Programming*, 1987.

[7] Ueda, K., Guarded Horn Clauses, *Proceedings of Logic Programming '85*, Lecture Notes in Computer Science 221, Springer-Verlag, Berlin, Heidelberg, 1986, pp.168-179.

APPENDICES

A. GHC Program

The GHC program of our algorithm is represented below. The initial goal of the program is *go(Ms)*, where *Ms* is a stream variable of lists of triplets, (*NodeName, Marker, Priority*). In this program, the priority of the *mm* is 0 for convenience.

```
%  gen(Ms,N) represents the structure of a network.
%  Ms is the input, a list of triplets, (NodeName, Marker, Priority).
%  N is the output, a list of pairs, (Marker, Priority), of each node.

gen(Ms,N) :- true |
        node(ag,Ms,Na,Pa,[]                  ,TMa,_  ),
        node(sm,Ms,Nb,Pb,[(TMa,[(+,Ne)])]   ,_  ,_  ),
        node(w ,Ms,Nc,Pc,[(TMa,[(+,Ng)])]   ,_  ,_  ),
        node(j ,Ms,Nd,Pd,[]                  ,TMd,FMd),
        node(nh,Ms,Ne,Pe,[(TMd,[]),(FMf,[])],_  ,_  ),
        node(as,Ms,Nf,Pf,[]                  ,TMf,FMf),
        node(sh,Ms,Ng,Pg,[(TMf,[]),(FMd,[])],_  ,_  ),
        N=[(Na,Pa),(Nb,Pb),(Nc,Pc),(Nd,Pd),(Ne,Pe),(Nf,Pf),(Ng,Pg)].

%  go(Ms) is the initial goal of the program.
%  Ms is the input, a stream of lists of triplets, (NodeName, Marker, Priority).

go(Ms) :- true | go1(Ms,Os) , outstream(Os).
go1([M1|Ms],Os) :- true | gen(M1,N) , Os=[nl,write(M1),nl,write(N),nl|Os1],
    go1(Ms,Os1).
go1([]     ,Os) :- true | Os=[write('terminated.'),nl].

%  node(Name,Ms,N,P,Xs,TM,FM) computes a marker, N, and a priority, P, of a
%  node corresponding to Name, and markers propagated from the node, using
%  input Ms, which is a list of triplets, (NodeName, Marker, Priority), and
%  Xs, which is a list of links to the node. TM (FM) is a marker to be
%  propagated through is-a links (is-not-a links, respectively) from the node.

node(Name,[(N1,_,_)|Cs],N,P,Xs,TM,FM) :- N1\=Name | node(Name,Cs,N,P,Xs,TM,FM).
node(Name,[],N,P,Xs,TM,FM) :- true |
    search(mm,0,1,1,Xs,N,P,MaxC) , prop(N,TM1,FM1) , C:=MaxC+1 , TM=(TM1,P,C),
    FM=(FM1,P,C).
node(Name,[(Name,M,P)|_],N,P1,Xs,TM,FM) :- true | search(M,P,0,0,Xs,N,P1,_),
    prop(N,TM1,FM1) , TM=(TM1,P1,1) , FM=(FM1,P1,1).

%  prop(M,TM,FM) chooses the types of markers that a node with marker M
%  propagates.

prop(tm,TM,FM) :- true  | TM=tm , FM=fm.
```

```
prop(M ,TM,FM) :- M\=tm | TM=mm , FM=mm.

%  search(TM,TP,TMax,TMin,Xs,N,P,C) computes a marker, N, a priority, P,
%  and a logical path length, C. TM, TP, TMax and TMin are a temporary
%  marker, a temporary priority, a temporary maximum logical path length, and
%  a temporary minimum logical path length, respectively. Xs is a list of
%  ((marker, priority, logical path length),constraint).

search(TM,TP,TMax,TMin,[((mm,_,_),_)|Xs],N,P,C) :- true |
    search(TM,TP,TMax,TMin,Xs,N,P,C).
search(mm,_ ,_   ,_   ,[((M1,P1,C1),Cons)|Xs],N,P,C) :- M1\=mm | con(Cons,CM),
    check(CM,mm,M1,NM,0,C1,NC,_,_,_,0,P1,NP) , search(NM,NP,NC,NC,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((_ ,P1,_),Cons)|Xs],N,P,C) :- TM\=mm,TP<P1 |
    search(TM,TP,TMax,TMin,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((M1,P1,C1),Cons)|Xs],N,P,C) :- M1\=mm,P1<TP |
    con(Cons,CM) , check(CM,TM,M1,NM,TMin,C1,NMin,TMax,C1,NMax,TP,P1,NP),
    search(NM,NP,NMax,NMin,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((M1,P1,C1),Cons)|Xs],N,P,C) :- TM\=mm,P1=TP,TMin=<C1 |
    max(TMax,C1,NMax) , search(TM,TP,NMax,TMin,Xs,N,P,C).
search(TM,TP,TMax,TMin,[((M1,P1,C1),Cons)|Xs],N,P,C) :- M1\=mm,P1=TP,TMin>C1 |
    con(Cons,CM) , check(CM,TM,M1,NM,TMin,C1,NMin,_,_,_,_,_,_),
    search(NM,TP,TMax,NMin,Xs,N,P,C).
search(TM,TP,TMax,_,[],N,P,C) :- true | C=TMax , N=TM , P=TP.

max(X,Y,Z) :- X>=Y | Z=X.
max(X,Y,Z) :- X<Y  | Z=Y.

%  con(Cons,CM) checks whether the constraint, Cons, is satisfied.

con([]        ,CM) :- true  | CM=tm.
con([(+,N)|Xs],CM) :- N=tm  | con(Xs,CM).
con([(-,N)|Xs],CM) :- N\=tm | con(Xs,CM).
con([(+,N)|Xs],CM) :- N\=tm | CM=fm.
con([(-,N)|Xs],CM) :- N=tm  | CM=fm.

%  check(CM,...) chooses a temporary marker, a temporary minimum logical path
%  length, a temporary maximum logical path length and a temporary priority.

check(tm,_ ,M2,M3,_    ,MinC2,MinC3,_    ,MaxC2,MaxC3,_ ,P2,P3) :- true |
    M3=M2 , MinC3=MinC2 , MaxC3=MaxC2 , P3=P2.
check(fm,M1,_ ,M3,MinC1,_    ,MinC3,MaxC1,_    ,MaxC3,P1,_ ,P3) :- true |
    M3=M1 , MinC3=MinC1 , MaxC3=MaxC1 , P3=P1.
```

B. Example

Suppose that we want to know, first, whether August is in the summer or in the winter in Japan, and second, whether August is in the summer or in the winter in

Australia. For the first question, we set a marker, *tm*, with priority 1 on node *ag*, corresponding to August, a marker, *tm*, with priority 2 on node *j*, corresponding to Japan. For the second question, we set a marker, *tm*, with priority 1 on node *ag*, a marker, *tm*, with priority 2 on node *as*, corresponding to Australia. Then our program runs as follows. Note that the two questions are also processed in parallel.

```
| ?- ghc go([[(ag,tm,1),(j,tm,2)],[(ag,tm,1),(as,tm,2)]]).

[(ag,tm,1),(j,tm,2)]
[(tm,1),(tm,1),(mm,0),(tm,2),(tm,2),(mm,0),(fm,2)]

[(ag,tm,1),(as,tm,2)]
[(tm,1),(mm,0),(tm,1),(mm,0),(fm,2),(tm,2),(tm,2)]

terminated.
```

In the first solution list, the second pair is (*tm*,1) and the second node, *sm*, means summer, so August is in the summer in Japan. Similarly, in the second solution list, the third pair is (*tm*,1) and the third node, *w*, means winter, so August is in the winter in Australia.